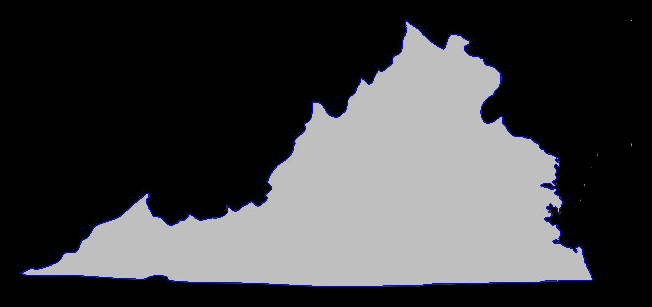
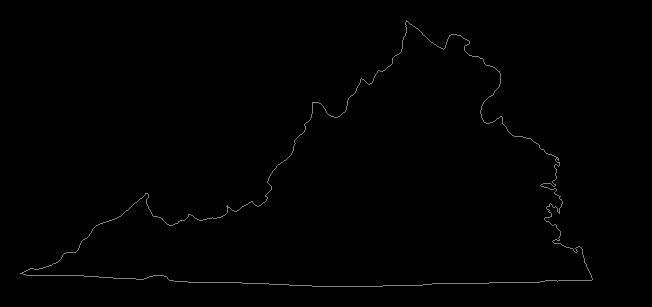
**Part 1**

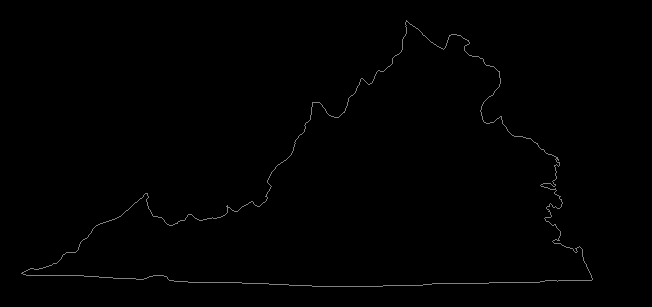
**Initial Contour**

****

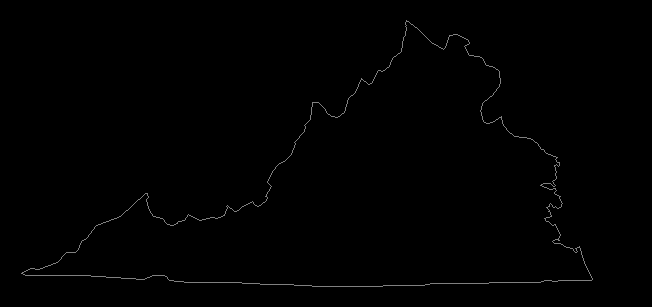
**Step 0 Contour**

****

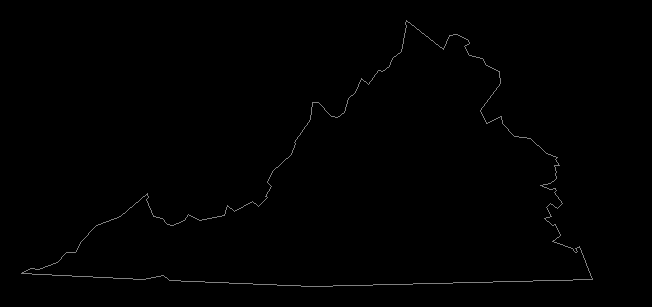
**Step 1 Contour**

****

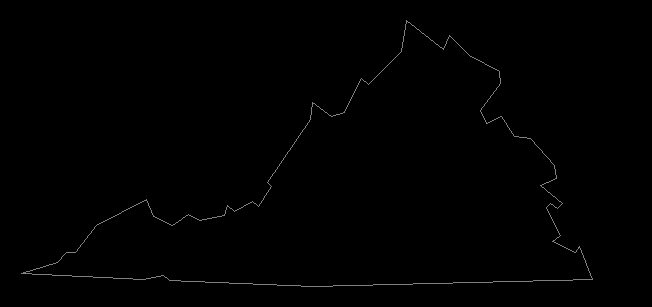
**Step 2 Contour**

****

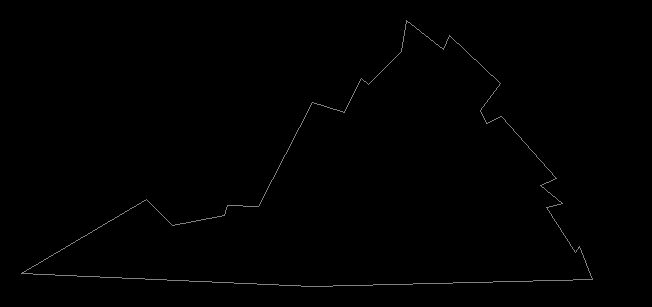
**Step 3 Contour**

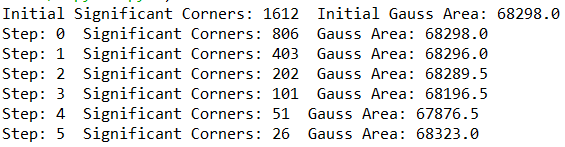
****

**Step 4 Contour**

****

**Step 5 Contour**





**Code Used:**

# -\*- coding: utf-8 -\*-

"""

ECE5554 FA19 HW3 part 1.py - contours

Created on Fri Oct 25 10:34:24 2019

@author: crjones4

"""

# CompVisHw3\_1.py - implemented Pavlidis contour Extraction

# Created on 11/12/19

# @Updated: Richard Ngo

import numpy as np

import cv2

import math

#pathName = "C:\\Data\\" # change this for your own file structure

pathName = ""

MAXCONTOUR = 5000

doLogging = False

def showImage(img, name):

cv2.imshow(name, img)

return

################################

def saveImage(img, name):

cv2.imwrite(name + ".png", img)

return

################################

def GaussArea(pts):

area = 0

for i in range(len(pts)):

if(i==len(pts)-1):

P1 = pts[i]

P2 = pts[0]

else:

P1 = pts[i]

P2 = pts[i+1]

area +=(P1[0]\*P2[1]-P1[1]\*P2[0])/2

return abs(area);

################################

def onePassDCE(ctrIn):

Kmin = 0

imin = 0

for i in range(len(ctrIn)):

if(i==0):

P1 = ctrIn[len(ctrIn)-1]

P2 = ctrIn[i]

P3 = ctrIn[i+1]

#print(P1,P2,P3)

elif(i==len(ctrIn)-1):

P1 = ctrIn[i-1]

P2 = ctrIn[i]

P3 = ctrIn[0]

else:

P1 = ctrIn[i-1]

P2 = ctrIn[i]

P3 = ctrIn[i+1]

#print(P1,P2,P3)

#print(range(len(ctrIn)))

# print(P1[1],P2[1],P3[1])

# print(P1[0],P2[0],P3[0])

L1 = ((P2[1]-P1[1])\*\*(2)+(P2[0]-P1[0])\*\*(2))\*\*(1/2)

L2 = ((P3[1]-P2[1])\*\*(2)+(P3[0]-P2[0])\*\*(2))\*\*(1/2)

if((P2[0]-P1[0]) == 0):

ang1 = (P2[1]-P1[1])/abs(P2[1]-P1[1])\*math.pi/2

else:

ang1 = (P2[1]-P1[1])/(P2[0]-P1[0])

if((P3[0]-P2[0]) == 0):

ang2 = (P3[1]-P2[1])/abs(P3[1]-P2[1])\*math.pi/2

else:

ang2 = (P3[1]-P2[1])/(P3[0]-P2[0])

angD = math.atan(ang1)-math.atan(ang2)

K = abs(angD\*L1\*L2/(L1+L2))

if(K<Kmin or i==0):

#print(K,i)

Kmin = K

imin = i

trimmedContour = np.append(ctrIn[0:imin],ctrIn[imin+1:len(ctrIn)], axis=0)

"""

contourImage = cv2.imread('VAoutline.png')

for i in range(len(trimmedContour)):

contourImage[trimmedContour[i,1],trimmedContour[i,0]] = [255,0,0]

cv2.namedWindow('contour', flags=cv2.WINDOW\_NORMAL)

cv2.imshow('contour', contourImage)

cv2.resizeWindow('contour', (int(len(contourImage[0])\*2), int(len(contourImage)\*2)))

cv2.waitKey(0)

cv2.destroyAllWindows()

"""

return trimmedContour

################################

def Pavlidis(img, start):

contourImage = cv2.imread('VAoutline.png')

stuck = 0

points = np.array([start])

trace = np.array([start[0],start[1]])

vert=1

hor=0

while 1:

i = 0

while i < 3:

horP = hor

vertP = vert

#print(img[trace[1]+(0-1)\*hor-1\*vert,trace[0]+(0-1)\*vert+1\*hor], img[trace[1]+(1-1)\*hor-1\*vert,trace[0]+(1-1)\*vert+1\*hor], img[trace[1]+(2-1)\*hor-1\*vert,trace[0]+(2-1)\*vert+1\*hor], trace[1],trace[0], vert, hor, i)

#if(img[trace[1]+(i-1)\*hor-1\*vert,trace[0]+(i-1)\*vert+1\*hor] != img[trace[1]-1\*hor,trace[0]-1\*vert]):

if(img[trace[1]+(i-1)\*hor-1\*vert,trace[0]+(i-1)\*vert+1\*hor] > 0):

#print(trace[1],trace[0], vert, hor)

stuck=0

trace[1]+=(i-1)\*hor-1\*vert

trace[0]+=(i-1)\*vert+1\*hor

contourImage[trace[1],trace[0]] = [255,0,0]

if(trace[0] == start[0] and trace[1] == start[1]):

cv2.namedWindow('contour', flags=cv2.WINDOW\_NORMAL)

cv2.imshow('contour', contourImage)

#print(points)

cv2.waitKey(0)

cv2.destroyAllWindows()

return points

points = np.append(points, [[trace[0],trace[1]]], axis=0)

#hor = (i-1)\*vertP+horP\*(2-i)\*(i)

#vert = -(i-1)\*horP+vertP\*(2-i)\*(i)

if(i!=2):

hor = (i-1)\*vertP+horP\*i

vert = -(i-1)\*horP+vertP\*i

i=-1

i+=1

stuck = stuck+1

if(stuck == 3):

points = np.append(points, [[trace[0],trace[1]]], axis=0)

return points

hor = vertP

vert = -horP

################################

def showContour(ctr, img, name):

contourImage = img

length = ctr.shape[0]

for count in range(length):

contourImage[ctr[count, 1], ctr[count, 0]] = 0

cv2.line(contourImage,(ctr[count, 0], ctr[count, 1]), \

(ctr[(count+1)%length, 0], ctr[(count+1)%length, 1]),(128,128,128),1)

showImage(contourImage, name)

saveImage(contourImage, name)

#################################

inputImage = cv2.imread(pathName + 'VAoutline.png', cv2.IMREAD\_GRAYSCALE)

thresh = 70;

binary = cv2.threshold(inputImage, thresh, 255, cv2.THRESH\_BINARY)[1]

(height, width) = binary.shape

# find a start point

ystt = np.uint8(height/2) # look midway up the image

for xstt in range(width): # from the left

if (binary[ystt, xstt] > 0):

break

contour = Pavlidis(binary, [xstt, ystt])

showContour(contour, inputImage, "CONTOUR")

print("Initial Significant Corners:",contour.shape[0], " Initial Gauss Area:",GaussArea(contour))

for step in range(6):

numLoops = math.floor(contour.shape[0]/2)

for idx in range(numLoops):

contour = onePassDCE(contour)

showContour(contour, np.zeros\_like(inputImage), "STEP"+str(step))

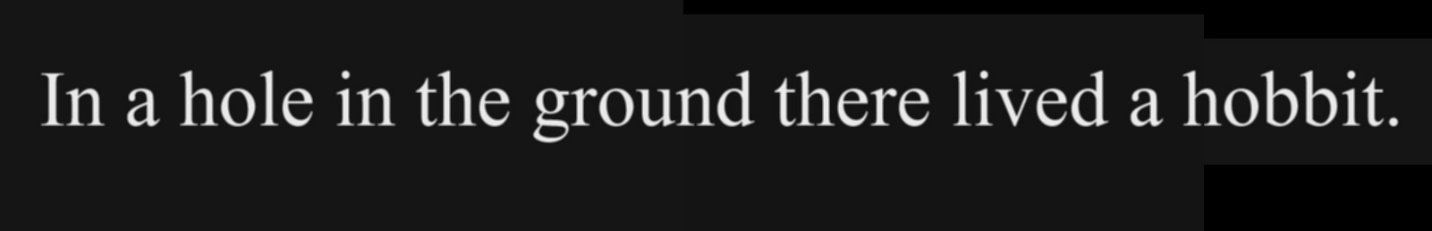
print("Step:",step," Significant Corners:",contour.shape[0], " Gauss Area:",GaussArea(contour))

cv2.waitKey(0)

cv2.destroyAllWindows()

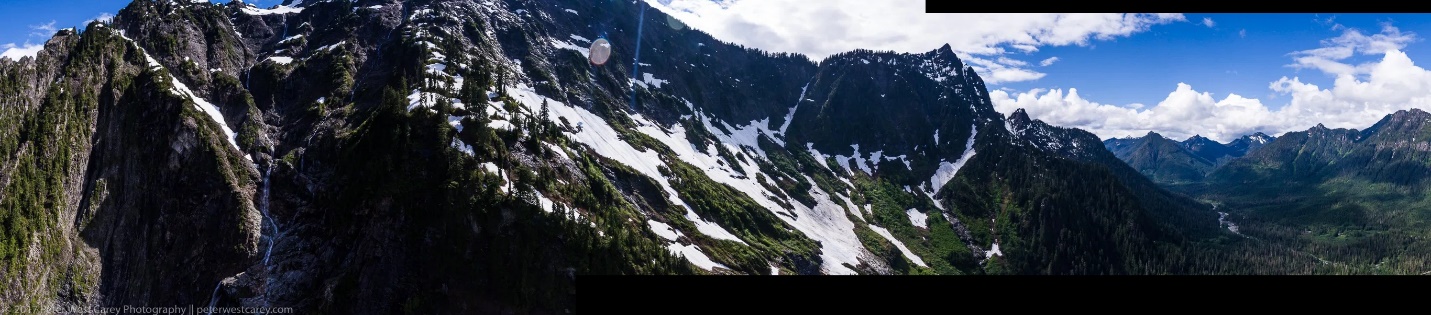
**Part 2**

**hobbit Aligned Images**

****

**Goodwin Aligned Images**

****

**BigFour Aligned Images**

**Code Used:**

# CompVisHw3\_1.py - Updated Code example code listed in lecture notes for image alignment

# https://www.learnopencv.com/image-alignment-feature-based-using-opencv-c-python/

# Created on 11/12/19

# @Updated: Richard Ngo

from \_\_future\_\_ import print\_function

import cv2

import numpy as np

MAX\_FEATURES = 500

GOOD\_MATCH\_PERCENT = 0.015

#############################################################

def showimg(img):

cv2.namedWindow('showimg', flags=cv2.WINDOW\_NORMAL)

cv2.imshow('showimg', img)

#cv2.resizeWindow('showimg', (int(len(img[0])\*2), int(len(img)\*2)))

#print(img)

cv2.waitKey(0)

cv2.destroyAllWindows()

#############################################################

def findMatches(im1, im2):

# Convert images to grayscale

im1Gray = cv2.cvtColor(im1, cv2.COLOR\_BGR2GRAY)

im2Gray = cv2.cvtColor(im2, cv2.COLOR\_BGR2GRAY)

# Detect ORB features and compute descriptors.

orb = cv2.ORB\_create(MAX\_FEATURES)

keypoints1, descriptors1 = orb.detectAndCompute(im1Gray, None)

keypoints2, descriptors2 = orb.detectAndCompute(im2Gray, None)

# Match features.

matcher = cv2.DescriptorMatcher\_create(cv2.DESCRIPTOR\_MATCHER\_BRUTEFORCE\_HAMMING)

matches = matcher.match(descriptors1, descriptors2, None)

# Sort matches by score

matches.sort(key=lambda x: x.distance, reverse=False)

# Remove not so good matches

numGoodMatches = int(len(matches) \* GOOD\_MATCH\_PERCENT)

matches = matches[:numGoodMatches]

# Draw top matches

imMatches = cv2.drawMatches(im1, keypoints1, im2, keypoints2, matches, None)

cv2.imwrite("matches.jpg", imMatches)

# Extract location of good matches

points1 = np.zeros((len(matches), 2), dtype=np.float32)

points2 = np.zeros((len(matches), 2), dtype=np.float32)

for i, match in enumerate(matches):

points1[i, :] = keypoints1[match.queryIdx].pt

points2[i, :] = keypoints2[match.trainIdx].pt

return points1, points2, matches[len(matches)-1].distance

#############################################################

def combineImages(points1, points2, im1,im2):

# Find homography

h, mask = cv2.findHomography(points1, points2, cv2.RANSAC)

print("here")

print(h)

# Use homography

height1, width1, channels1 = im1.shape

height2, width2, channels2 = im2.shape

if(h[0,2]<=0):

h1 = np.copy(h)

h1[0,2] = 0

h2=np.identity(3)

h2[0,2]=-h[0,2]

width = width2

else:

h1 = np.copy(h)

h2=np.identity(3)

width = width1

part1 = cv2.warpPerspective(im1, h1, (width+int(round(abs(h[0,2]))), height2))#width+math.ceil(h2[0,2])

part2 = cv2.warpPerspective(im2, h2, (width+int(round(abs(h[0,2]))), height2))

#showimg(part1)

#showimg(part2)

#imReg = np.zeros(shape=(height,width))

"""

if(h1[0,2]==0):

#imReg[:math.floor(h2[0,2])] = part1[:math.floor(h2[0,2])]

imReg=part1

imReg[:,int(round(h2[0,2])):,:] = part2[:,int(round(h2[0,2])):,:]

else:

#imReg[:math.floor(h1[0,2])] = part2[:math.floor(h1[0,2])]

imReg=part2

imReg[:,int(round(h1[0,2])):,:] = part1[:,int(round(h1[0,2])):,:]

"""

"""

if(h1[0,2]==0):

#imReg[:math.floor(h2[0,2])] = part1[:math.floor(h2[0,2])]

imReg=part1

imReg[part2>0] = part2[part2>0]

else:

#imReg[:math.floor(h1[0,2])] = part2[:math.floor(h1[0,2])]

imReg=part2

imReg[part1>0] = part1[part1>0]

"""

imReg=part1

imReg[part2>0] = part2[part2>0]

#showimg(imReg)

return imReg, h

#############################################################

def alignImages(im1in, im2in, im3in):

#sorter = [im1in,im2in,im3in]

#sorter.sort(key=lambda x: len(x), reverse=True)

#[im1, im2, im3] = sorter

points12, points21, MD12 = findMatches(im1in,im2in)

points13, points31, MD13 = findMatches(im1in,im3in)

points21, points12, MD21 = findMatches(im2in,im1in)

points23, points32, MD23 = findMatches(im2in,im3in)

points31, points13, MD31 = findMatches(im3in,im1in)

points32, points23, MD32 = findMatches(im3in,im2in)

compad = [MD12+MD13,MD21+MD23,MD31+MD32]

center = compad.index(min(compad))

if(center == 0):

if(MD12<MD13):

[im1,im2,im3]=[im2in,im1in,im3in]

[points1,points2] = [points21, points12]

else:

[im1,im2,im3]=[im3in,im1in,im2in]

[points1,points2] = [points31, points13]

elif(center == 1):

if(MD21<MD23):

[im1,im2,im3]=[im1in,im2in,im3in]

[points1,points2] = [points12, points21]

else:

[im1,im2,im3]=[im3in,im2in,im1in]

[points1,points2] = [points32, points23]

else:

if(MD31<MD32):

[im1,im2,im3]=[im1in,im3in,im2in]

[points1,points2] = [points13, points31]

else:

[im1,im2,im3]=[im2in,im3in,im1in]

[points1,points2] = [points23, points32]

#[im1,im2]=[np.copy(im2),np.copy(im1)]

#[points1,points2]=[np.copy(points2),np.copy(points1)]

#[im1,im3]=[np.copy(im3),np.copy(im1)]

#points1, points2, matches = findMatches(im1,im2)

"""

print(matches2[len(matches2)-1].distance, len(matches2))

print("here")

print(matches3[len(matches3)-1].distance, len(matches3))

if(matches2[len(matches2)-1].distance<=matches3[len(matches3)-1].distance):

"""

halfReg, h1= combineImages(points1, points2,im1,im2)

points1, points2, matches3 = findMatches(halfReg,im3)

fullReg, h2 = combineImages(points1, points2,halfReg,im3)

"""

else:

halfReg, h1 = combineImages(points13, points31,im1,im3)

points1, points2, matches2 = findMatches(halfReg,im2)

fullReg, h2 = combineImages(points1, points2,halfReg,im2)

"""

#showimg(fullReg[:,:(width1+width2+width3-abs(math.floor(h1[0,2]))-abs(math.floor(h2[0,2]))),:])

return fullReg, h1, h2

#############################################################

if \_\_name\_\_ == '\_\_main\_\_':

# # Read images to be aligned

ImgNames = ["hobbit","goodwin","BigFour"]

for I in ImgNames:

im1R = I+"0.png"

im2R = I+"1.png"

im3R = I+"2.png"

print("Reading ", im1R)

im1 = cv2.imread(im1R, cv2.IMREAD\_COLOR)

print("Reading ", im2R)

im2 = cv2.imread(im2R, cv2.IMREAD\_COLOR)

print("Reading ", im3R)

im3 = cv2.imread(im3R, cv2.IMREAD\_COLOR)

print("Aligning images ...")

# Registered image will be resotred in imReg.

# The estimated homography will be stored in h.

imReg, h1, h2 = alignImages(im1, im2, im3)

# Write aligned image to disk.

outFilename = I+"\_aligned.jpg"

print("Saving aligned image : ", outFilename);

cv2.imwrite(outFilename, imReg)

# Print estimated homography

print("Estimated homographies for: "+I+"\n", h1)

print(h2)